Click Fraud Detection and Prevention System For Ad Networks

Paulo S. de Almeida[1], **João J. C. Gondim**[1]

[1]Departamento de Ciência da Computação – Universidade de Brasília (UnB)
Brasília – DF – Brazil

Abstract. *Click fraud detection consists of identifying the intention behind the clicks a system receives. A system that detects and prevents this type of fraud is proposed and implemented, based on the ad network, one of the 3 agents in the online ad environment. To validate, 3 servers were used, representing said agents. A bot simulates an attacker, and the frauds are ultimately identified by our proposed detector in tested scenarios.*

1. Introduction

The publicity domain grows continually by the day. The report from *eMarketer* [McNair 2017] states an estimated growth of nearly 16% for digital advertisement in the United States compared to the previous year, almost 83 billion dollars (roughly 40% of the overall investment in advertising). With these numbers, we can see the importance of the domain and related fields, like detection of click frauds, which are essentially clicks generated by malicious users. Their identification relies heavily on HTTP requests and contextual information for such requests.

2. Technical Review

Online advertisement involves four agents. The **advertiser** wishes to publicize their product or service to people that may be interested, the **users**. To do that, they count with the **publisher**, someone with their own publicity platform. The **ad network** is a middleman between publishers and advertisers. Click fraud happens in context of a pay-per-click model, where the advertiser pays for every click their ad receives, and can be defined as intentional clicks, automated or done by a person, that don't represent interest in the ad.

[Kitts et al. 2015] presents an overview on the *Microsoft adCenter* system and design aspects of it. [Xu et al. 2014] provides a report on the techniques used for a fraud detection system on the advertiser. However, the lack of reports on ad network systems is evident, and the elaboration upon design details and implementation of fraud detection leaves something to be desired. On fraud methods, [Daswani and Stoppelman 2007] reports on the inner workings of a click fraud *botnet*, and presents the concept of low-frequency attacks, which occur over a long period of time, using a small amount of clicks over short time slots, as to not be detected by methods that look at expected click rates for a given ad. Reports such as [Leyden 2006] and [Court 2009] go over other cases.

3. Proposal and Implementation

[Kitts et al. 2015]'s proposal is the basis for the overall architecture of the system. The rules **JavascriptEnabledRule** and **ExternalBehaviorRule** in the system derive from

[Xu et al. 2014], while the other rules and their classifications are this work's novel contribution.

The system has two parallel processes, **Online Analysis** and **Offline Analysis**. In the former, the user goes through a module called **URL Hash**, which will prevent reuse of ad URLs by making a unique URL for the user when they see the ad, based on HTTP headers and a private key from the system, and then verifies it when the user clicks the ad. Afterwards the user gets redirect to two pages in the domain as their requests go through the *online rules*. Finally, the system's analysis on the user's legitimacy is stored in the **Database**. The latter process takes those recorded requests as input and does further analysis on them through the *offline rules*, being the last part of the automatic process to judge the users' intentions.

Rules are the main method through which the system will detect frauds, and every click the ad receives passes through all the rules. They can be divided between **online rules**, which are quick and executed during the user's access to the ad network's domain and **offline rules**, which use the request database to identify odd access behavior from IPs. Two more groups are defined for the rules: **decisive rules** will identify a user as being fraudulent if they fail at any of them, while **indicative rules** give a probability for the user to be malicious. The latter have weights associated with them, which are rational numbers, including negatives. Negative weight indicates that failing the rule will not hurt the user's chance of being considered legitimate.

Decisive and online rules are **BlacklistRule** (checks if the user's IP is in a table of blacklisted IP addresses), **HumanTimerRule** (verifies if the time between the ad loading or different clicks from the same user are humanly possible, given the average human reaction time of roughly 0.2 seconds [Census at School Canada 2017], [Thorpe et al. 1996], and 0.3 seconds estimated for the user to be interested enough to click) and **AcceptLangRule** (looks for a valid *Accept-Language* header). **PagesLoadedRule** is decisive and offline, and verifies if the user has visited all the expected pages, like javascript files, while not accessing pages a normal user wouldn't, like URLs hidden in HTML code.

Indicative and online rules are **JavascriptEnabledRule** (most internet users have javascript and cookies enabled [Xu et al. 2014], [Priebe 2009], [Winnicki 2016]. This rule will thus verify both by sending a the user a cookie via javascript code in the page), **UserAgentRule** (verifies if the request contains a valid UserAgent HTTP header), **DoNotTrackRule** (confirms if the user's request has a DNT field. Should have a *negative weight*, as explained previously) and **RedirectTimeRule** (looks at how much time the user took to be redirected from the 1st to the 2nd page. A higher delay indicates the user might be a bot).

Indicative and offline rules are **TimePeriodRule** (looks over the stored requests to find odd access patterns by searching for 3 clicks within a short period or 5 or more clicks within a longer period, all from the same IP) **ExternalBehaviorRule** (verifies reports sent by advertisers. The report should contain categories like **Number of Clicks** and **Mouse Events**, split by the values in the first page and in other pages).

During the click process, the user will be redirect through 2 pages in the ad network's domain before being sent to the ad's page; this is done to obtain information that wouldn't be possible from a single page access. The first page will put the request through

the **HumanTimer**, **Blacklist**, **AcceptLang** and **DNT** rules, while the 2nd will aply the rules **UserAgent**, **Javascript** and **RedirectTime**.

The attack bot created to test the system focuses on getting the highest number of clicks possible without being caught. To emulate the progress a real attacker would go through, the bot has different **modules**, which alter how the bot goes through with its attacks, and they are **NormalAccessModule** (bot obtains only valid ad addresses), **HumanTimerModule** (the program waits a humanly possible time before executing each click), **HeadersModule** (uses legitimate HTTP headers to imitate a user with a browser), **DNTModule** (adds a valid DoNotTrack header to the requests), **WideAccessModule** (the bot will access every link passed by the pages received), **SelectiveAccessModule** (extends on *WideAccessModule* so the bot doesn't access links a normal user wouldn't, such as images hidden in the HTML), **RandomTimeModule** (adds a pseudo-random amount of time between the clicks), **CookieModule** (bot stores cookies sent by the pages correctly) and **CompleteAccessModule** (extends on *SelectiveAccessModule*, loads pages in ways that are expected from a user, such as loading images and page icons).

4. Results

To test the system we first created locally hosted sites for the 3 online advertisement agents and selected a few configurations of modules for the bot, 2 of which are reported below. The bot went through the click process 3 times for each. Finally, we check to see if the system detected the invalid clicks by defining the score for passing through the tests to be *0.5 or higher*. This process was done in a Windows 10 operating system. The weights used are **2.0** for *JavascriptEnabledRule*, *UserAgentRule* and *TimePeriodRule*, **3.0** for *BehaviorRule* and *RedirectTimeRule*, and **-1.0** for *DNTRule*.

The first part of the test was to see if the system would incorrectly label a real user as a fraud. Sites were accessed normally with the Firefox 59.0.1 (64-bit) browser. Table 1 shows results for every rule test the click went through. A value of 't' means success, while 'f' means failure, and the real user passed through all online tests without issues. In the following test, we use the bot with the following modules used: *NormalAccess*, *HumanTimer*, *Headers*, *DNT* and *WideAccess*. The bot does not pass the tests, as in Table 2, thanks to rules *JavascriptEnabled* and *RedirectTime*. For the final test, the bot has every module activated. The system still manages to detect the fraud, thanks to the rules *RedirectTime* and *TimePeriod*, with the results reported in Table 3.

Table 1. System results for a real user

id	*dnt*	*java*	*human*	*list*	*agent*	*redir*	*lang*	*behav*	*pages*	*tperiod*	*score*
2	t	t	t	t	t	t	t				1.17

Table 2. System results for the bot configuration

id	*dnt*	*java*	*human*	*list*	*agent*	*redir*	*lang*	*behav*	*pages*	*tperiod*	*score*
9	t	f	t	t	t	f	t				0.43

5. Conclusion

Results obtained match expectations. The system, although not yet tested in a real environment, has shown good performance against different attacks, with at least two of

Table 3. System results for the final bot configuration

id	*dnt*	*java*	*human*	*list*	*agent*	*redir*	*lang*	*behav*	*pages*	*tperiod*	*score*
18	t	t	t	t	t	f	t		t	f	0.42

the rules identifying each of those attempts. All the attacks attempted by the bot were identified, but it is emphasized the importance of selecting appropriate values for rules' weights. Low-frequency attacks are still a threat as malicious clicks spread over long time intervals will not be detected by the system; there are, however, issues with those methods of attack, namely obtaining access to this number of IPs and obtaining economical return from simpler attempts of the attack, given the low return for a single click.

References

Census at School Canada (2017). Average reaction time. `http://censusatschool.ca/data-results/2016-2017/average-reaction-time/`.

Clifford, S. (2009). Microsoft sues three in click-fraud scheme.

Court, U. S. D. (2009). Microsoft vs Eric Lam et. al.

Daswani, N. and Stoppelman, M. (2007). The anatomy of clickbot. a. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 11–11. USENIX Association.

Fielding, R. and Reschke, J. (2014). Hypertext transfer protocol (http/1.1): Semantics and content. RFC 7231, RFC Editor. `http://www.rfc-editor.org/rfc/rfc7231.txt`.

Human Benchmark (2018). Reaction time statistics. `https://www.humanbenchmark.com/tests/reactiontime/statistics`.

Kitts, B., Zhang, J. Y., Wu, G., Brandi, W., Beasley, J., Morrill, K., Ettedgui, J., Siddhartha, S., Yuan, H., Gao, F., et al. (2015). Click fraud detection: adversarial pattern recognition over 5 years at microsoft. In *Real World Data Mining Applications*, pages 181–201. Springer.

Leyden, J. (2006). Botnet implicated in click fraud scam.

McNair, C. (2017). Us ad spending: emarketer's updated estimates and forecast for 2017.

Oentaryo, R. J., Lim, E.-P., Finegold, M., Lo, D., Zhu, F., Phua, C., Cheu, E.-Y., Yap, G.-E., Sim, K., Nguyen, M. N., et al. (2014). Detecting click fraud in online advertising: a data mining approach. *Journal of Machine Learning Research*, 15(1):99–140.

Priebe, J. (2009). A study of internet users' cookie and javascript settings.

Thorpe, S., Fize, D., and Marlot, C. (1996). Speed of processing in the human visual system. *nature*, 381(6582):520.

Winnicki, A. (2016). Just how many web users really disable cookies or javascript?

Xu, H., Liu, D., Koehl, A., Wang, H., and Stavrou, A. (2014). Click fraud detection on the advertiser side. In *European Symposium on Research in Computer Security*, pages 419–438. Springer.